Admin

- Midterm Thursday
  - Review question sessions tonight and Wednesday

- Assignment 3?

- Assignment 4 out soon
  - Due Monday 10/12

Examples from this lecture

http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/cs41b/

CS41B machine

CPU

- Instruction counter (location in memory of the next instruction in memory)
  - holds the value 0 (read only)

- Processor

- General purpose
  - read/write

- Registers
  - r0
  - r1
  - r2
  - r3
CS41B code

Four main types of operations
1. math
2. branch/conditionals
3. memory
4. control the machine (e.g. stop it)

CS41B execution

More specifically, the CS41B Machine cycles through the following steps:
- The machine fetches the value at mem[1c] for use as an instruction.
- The machine increments the value in 1c by 2.
- The machine decodes and carries out the instruction.

Notice that, while an instruction is being executed, the value in 1c is the address of the next instruction. This detail is important in implementing branch instructions.

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>RR-</td>
<td>dest = src0</td>
</tr>
<tr>
<td>neg</td>
<td>RR-</td>
<td>dest = -src0</td>
</tr>
<tr>
<td>add</td>
<td>RRR</td>
<td>dest = src0 + src1</td>
</tr>
<tr>
<td>sub</td>
<td>RRR</td>
<td>dest = src0 - src1</td>
</tr>
<tr>
<td>adc</td>
<td>RRS</td>
<td>dest = src0 + arg</td>
</tr>
<tr>
<td>sbc</td>
<td>RRS</td>
<td>dest = src0 - arg</td>
</tr>
</tbody>
</table>

operation name
(always three characters)
<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>RR-</td>
<td>dest = src0</td>
</tr>
<tr>
<td>neg</td>
<td>RR-</td>
<td>dest = -src0</td>
</tr>
<tr>
<td>add</td>
<td>RRR</td>
<td>dest = src0 + src1</td>
</tr>
<tr>
<td>sub</td>
<td>RRR</td>
<td>dest = src0 - src1</td>
</tr>
<tr>
<td>adc</td>
<td>RRS</td>
<td>dest = src0 + arg</td>
</tr>
<tr>
<td>sbc</td>
<td>RRS</td>
<td>dest = src0 - arg</td>
</tr>
</tbody>
</table>

**operation arguments**
R = register (e.g. r0)
S = signed number (byte)

**operation function**
dest = first register
src0 = second register
src1 = third register
arg = number/argument

---

**Register Instructions**

```
adc r1 r0 8           r1 = 8
neg r2 r1            r2 = -8, r1 = 8
sub r2 r1 r2         r2 = 16
```

```
adc r1 r0 8           r1 = 8
neg r2 r1            r2 = -8, r1 = 8
sub r2 r1 r2         r2 = 16
```
<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sto</strong></td>
<td>RR[5]</td>
<td>mem(dest + arg) − src0</td>
</tr>
<tr>
<td><strong>loa</strong></td>
<td>RR[5]</td>
<td>dest = mem(src0 + arg)</td>
</tr>
</tbody>
</table>

**sto** = save data in register TO memory

**loa** = put data FROM memory into a register

**Special cases:**
- saving TO (sto) address 0 prints
- reading from (loa) address 0 gets input from user

---

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nop</strong></td>
<td>---</td>
<td>do nothing</td>
</tr>
<tr>
<td><strong>hlt</strong></td>
<td>---</td>
<td>stop the machine</td>
</tr>
<tr>
<td><strong>pau</strong></td>
<td>---</td>
<td>pause the machine</td>
</tr>
</tbody>
</table>

---

**Basic structure of CS41B program**

```plaintext
; great comments at the top!

; instruction1       ; comment
instruction2         ; comment
...
hlt                   
end                   
```

whitespace before operations/instructions

---

**Running the CS41B machine**

- Look at subtract.a41
- load two numbers from the user
- subtract
- print the result
CS41B simulator

Different windows
- Memory (left)
- Instruction execution (right)
- Registers
- I/O and running program

Modify ic, the instruction counter... which changes the flow of the program!

What do these operations do?

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>beq</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>bne</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>bgt</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>bge</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>blt</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>bge</td>
<td>RRS</td>
<td></td>
</tr>
</tbody>
</table>

What does this do?

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>beq</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>bne</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>bgt</td>
<td>RRS</td>
<td></td>
</tr>
<tr>
<td>bge</td>
<td>RRS</td>
<td></td>
</tr>
</tbody>
</table>
beq r3 r0 done
If r3 = 0, branch to the label "done"
if not (else) ic is incremented as normal to
the next instruction

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>brs</td>
<td>--5</td>
<td>ic = loc + arg</td>
</tr>
<tr>
<td>beq</td>
<td>RRS</td>
<td>if dest = src0, ic = loc + arg</td>
</tr>
<tr>
<td>bne</td>
<td>RRS</td>
<td>if dest ≠ src0, ic = loc + arg</td>
</tr>
<tr>
<td>bgt</td>
<td>RRS</td>
<td>if dest &gt; src0, ic = loc + arg</td>
</tr>
<tr>
<td>ble</td>
<td>RRS</td>
<td>if dest ≤ src0, ic = loc + arg</td>
</tr>
<tr>
<td>bge</td>
<td>RRS</td>
<td>if dest ≥ src0, ic = loc + arg</td>
</tr>
</tbody>
</table>

ble r2 r3 done
If r2 <= r3, branch to the label done

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>arguments</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>brs</td>
<td>--5</td>
<td>ic = loc + arg</td>
</tr>
<tr>
<td>beq</td>
<td>RRS</td>
<td>if dest = src0, ic = loc + arg</td>
</tr>
<tr>
<td>bne</td>
<td>RRS</td>
<td>if dest ≠ src0, ic = loc + arg</td>
</tr>
<tr>
<td>bgt</td>
<td>RRS</td>
<td>if dest &gt; src0, ic = loc + arg</td>
</tr>
<tr>
<td>ble</td>
<td>RRS</td>
<td>if dest ≤ src0, ic = loc + arg</td>
</tr>
<tr>
<td>bge</td>
<td>RRS</td>
<td>if dest ≥ src0, ic = loc + arg</td>
</tr>
</tbody>
</table>

- Conditionals
- Loops
- Change the order that instructions are executed
Basic structure of CS41B program

; great comments at the top!
;
    instruction1 ; comment
    instruction2 ; comment
...
label1
    instruction ; comment
    instruction ; comment
label2...
    ... hlt
    end

- whitespace before operations/instructions
- labels go here

More CS41B examples

Look at max_simple.a41
- Get two values from the user
- Compare them
- Use a branch to distinguish between the two cases
  - Goal is to get largest value in r3
  - print largest value

What does this code do?

if( r3 < 0 ){ r2 = -1 }
else if( r3 != 0 ){ r2 = 1 }
else{ r2 = 0 }

bge r3 r0
elif sbc r2 r0 1
    brs endif
elif beq r3 r0
    adc r2 r0 1
    brs endif
else add r2 r0 r0
    brs endif
sto r0 r2
    hlt
    end

What does this code do?

bge r3 r0 endif
elif sbc r2 r0 1
    brs endif
elif beq r3 r0
    adc r2 r0 1
    brs endif
else add r2 r0 r0
    brs endif
sto r0 r2
    hlt
    end
What does this code do?

```assembly
bge r3 r0 1; if r3 >= 0 go to 1
adc r3 r0 1; r3 < 0; r2 = -1
bns endif; jump to end of if/elif/else

elif
 beq r3 r0 else; if r3 = 0 go to else
adc r2 r0 1; r3 > 0; r2 = 1
bns endif; jump to end of if/elif/else

else
 add r2 r0 r0; r3 = 0; r2 = 0
endif
sto r0 r2; print out r2

end
```

Your turn 😊

Write some code that prints out `abs(r3)`

```

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
<th>Func</th>
</tr>
</thead>
<tbody>
<tr>
<td>bge</td>
<td>r3</td>
<td>r0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>adc</td>
<td>r3</td>
<td>r0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>bns</td>
<td>endif</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td>r3</td>
<td>r0</td>
<td>else</td>
<td></td>
</tr>
<tr>
<td>adc</td>
<td>r2</td>
<td>r0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>bns</td>
<td>endif</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>r2</td>
<td>r0</td>
<td>r0</td>
<td></td>
</tr>
<tr>
<td>sto</td>
<td>r0</td>
<td>r2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Memory layout

- **Code**: Where program code is stored.
- **Heap**: Where dynamically allocated program data is stored.
- **Stack**: Where program/function execution information is stored, parameters, and local variables.
Stacks

Two operations
- push: add a value in the register to the top of the stack
- pop: remove a value from the top of the stack and put it in the register

<table>
<thead>
<tr>
<th>psh</th>
<th>R--</th>
<th>push the value in dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop</td>
<td>R--</td>
<td>pop the top of stack into dest</td>
</tr>
</tbody>
</table>

Stack frame

Key unit for keeping track of a function call
- return address (where to go when we’re done executing)
- parameters
- local variables

CS41B function call conventions

- r1 is reserved for the stack pointer
- r2 contains the return address
- r3 contains the first parameter
- additional parameters go on the stack (more on this)
- the result should go in r3

Structure of a single parameter function

```
frame
  psh r2 ; save return address on stack
  ... ; do work using r3 as argument
  ... ; put result in r3
  pop r2 ; restore return address from stack
  jmp r2 ; return to caller
```

conventions:
- argument is in r3
- r1 is off-limits since it’s used for the stack pointer
- return value goes in r3
Our first function call

```
loa r3 r0          ; get variable
loa r2 increment  ; call increment
cal r2 r2
sto r0 r3          ; write result,
hlt                ; and halt
```

increment
```
psh r2
adc r3 r3 1        ; add 1 to the input parameter
pop r2
jmp r2
```

Stack
```
sp (r1)
```

Our first function call

```
loa r3 r0
```

```
loa r2 increment
cal r2 r2
sto r0 r3
hlt
```

increment
```
psh r2
adc r3 r3 1
pop r2
jmp r2
```

Stack
```
sp (r1)
```

10/6/15
Our first function call

1. Go to instruction address in r2 (2nd r2)
2. Save current instruction address in r2

Stack

increment

sp [r1]
Our first function call

```plaintext
loa r3 r0
lcw r2 increment
cal r2 r2
sto r0 r3
hit

increment
psh r2
adc r3 r3 1
pop r2
jmp r2

Stack
```

Our first function call

```plaintext
loa r3 r0
lcw r2 increment
cal r2 r2
sto r0 r3
hit

increment
psh r2
adc r3 r3 1
pop r2
jmp r2

loc cal

Stack
```

Our first function call

```plaintext
loa r3 r0
lcw r2 increment
cal r2 r2
sto r0 r3
hit

increment
psh r2
adc r3 r3 1
pop r2
jmp r2

loc cal

Stack
```

Our first function call

```plaintext
loa r3 r0
lcw r2 increment
cal r2 r2
sto r0 r3
hit

increment
psh r2
adc r3 r3 1
pop r2
jmp r2

loc cal

Stack
```
Our first function call

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

```
loa r3 r0
loa r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```
Our first function call

```
loa r3 r0
lcw r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

Stack

```
loa r3 r0
lcw r2 increment
cal r2 r2
sto r0 r3
hit
increment
psh r2
adc r3 r3 1
pop r2
jmp r2
```

Stack

Examples from this lecture

http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/cs41b/